Medical Image Registration and Applications

Lab 1: Intensity based Image Registration

27 September, 2019

Abstract

Given 4 example images of the brain and a basic working framework for registration, we developed an enhanced version of the original system that is more rich in metrics, performs better and in a more robust way. Furthermore, we expanded the original data set in order to test our system. All of the combinations of all images were correctly registered.

Isaac Llorente Saguer

Pierpaolo Venditelli

MSc. Medical Imaging and Applications (MaIA)
Universitat de Girona

Contents

1 Questions

1.1 Registration Framework

Read and understand the Matlab files provided and answer the following issues: Identify each of the components of an image registration framework, state their type and where they can be found (file name and approximate line number).

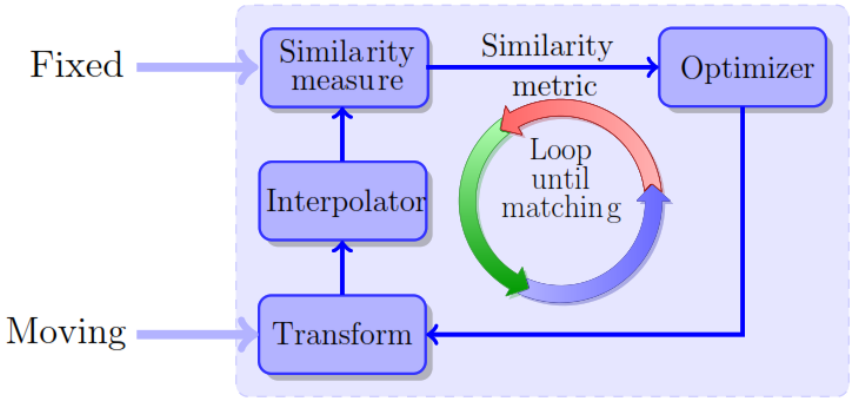

Figure 1: The framework used for image registration

The goal, is to modify the **"Moving Image"** so that it matches the **"Fixed Image"**. These two elements can be found in the code in the ***affineReg2D.m*** file [approx line 12-13].

Transformation Once the two imags are loaded, a set of transformations is applied to the **"Moving Image"**. In the scheme, this is the "Transform" block: In our case, we can choose either a "rigid" transformation or "affine" (but there exist other methods as well, such as b-splines). As well as the load of the images, these elements of the transform block are located in the same file (***affineReg2D.m***)[lines 28-43] [line 75 for applying the transformation].

Similarity Measure It represent a "score" of how much the two images are similar, according to the metric used. It is the value that the optimizer (next block) tries to minimize/-maximize. In the code it can be found in *"affine_registration_function"*, which is called by ***affineReg2D.m*** [line 17]. The function is a separate file(***affine_registration_function.m***). There are different types of metrics: for this lab **sd** (Square root of differences) was already implemented, while we implemented **cc** (Normalized Cross Correlation Intensity Based), **gcc** (Normalized Cross Correlation Intensity Based) and some more as we will see later.

Optimiser If we can somehow measure the similarity between two images, the optimiser can try to iteratively pick a combination of parameters that (hopefully) give the best results. The one used in the original files provided to us is *"fminsearch"*, built-in in MatLab. In the code, The optimiser function is called. ***affineReg2D*** [line 31]. This optimizer is tipically an algorithm based on gradient descend or similar. it iteratively updates the parameters of the transformation to reach the global optima of the metric.

Interpolator When transforming an image, not always we find 1:1 match between the image before and after the transformation. Thus, to ensure a smooth transformation, interpolation is commonly used and its function is to find ambiguus values by interpolating neighbour pixels. The file ***"image_ interpolation"*** is used for that purpose, and provides different ways of doing it (nearest, linear or cubic). Refer to *affineaffine_ transform_ 2d_ double*[line 75]

1.2 Scale vector

The function of the scale vector is to **adjust the weight of the parameters**. Since the expected parameters that will be used in the registration are of different order of magnitudes, levelling them out will help with the optimiser.

1.3 Center of Rotation

The center of rotation is the center of the image, since in the function *affine_ transform_ 2d_ double* it is made the (0,0) before applying the image transformation.

2 Similarity Metric

Add a new similarity metric to the framework: normalised cross-correlation(NCC). Implement the NCC metric for intensity and gradient (NCC and gradient NCC).

In this section we implemented the metric as requested by following the provided paper [1]. We developed a function called *calculatemetric* which is included in the file ***calculatemetric.m*** and the following code is the implementation of intensity and gradient NCC.

```
1       case 'cc'
2           a = (I_mov- mean(mean(I_mov)));
3           b = (I_fix- mean(mean(I_fix)));
4           numerator = sum(sum(a.*b));
5           d = sum(sum(a.^2));
6           d1 = sum(sum(b.^2));
7           denominator = sqrt(d*d1);
8           e =1-( numerator/denominator);
```

Listing 1: Normalized Cross Correlation Intensity Based

```
1       case 'gcc'
2           A1 = I_mov(1:end-1,2:end) - I_mov(1:end-1,1:end-1); %horizontal ...
                gradient
3           B1 = I_fix(1:end-1,2:end) - I_fix(1:end-1,1:end-1); %horizontal ...
                gradient fixed img
4           A = A1.*B1;
5           A2 = I_mov(2:end,1:end-1) - I_mov(1:end-1,1:end-1); %vertical ...
                gradient
6           B2 = I_fix(2:end,1:end-1) - I_fix(1:end-1,1:end-1); %vertical ...
                gradient fixed img
7           B = A2.*B2;
8           num = abs(A+B);
9           numerator = sum(sum(num));
10          d = sum(sum(A1.^2+A2.^2));
11          d1 = sum(sum(B1.^2+B2.^2));
12          denominator = sqrt(d*d1);
13          e =1-(numerator/denominator);
```

Listing 2: Normalized Cross Correlation Gradient Based

Other additional metrics were implemented and are provided in the code.

3 Transformation. Affine transformation

Modify the framework to be able to deal with full affine 2D transformations. Justify how you initialise the transformation.

In order to work with affine transformations, we have to expand the parameters , from 4 to 6 and work with homogeneous coordinates. The initialization we chose was the the one that translated into the identity matrix (refer to *Figure* 2), so as to start with the original moving image. Thus, the translation and shear are set to zero.

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Figure 2: Initialisation equal to the identity transformation

4 Multi-Resolution

Implement the above modifications in a multi-resolution registration framework. The number of resolutions should be a parameter of the framework, discuss how each resolution is initialised. Justify the benefits of multi-resolution in terms of computation time and/or accuracy of the final result, compared to a single resolution

Multi resolution is a useful technique to register two images. It works iteratively, first scaling by a factor of N the images and then applying the transformation and the optimizer to reach the optima. This is repeated for all the layers (level of scaling) from the smallest to the biggest. The way we implemented the code, the biggest layer would correspond to the original image. Then, the next one would be a lineal scale of $\frac{1}{2}$, so the information for the modified image is easily taken by averaging 2 pixels. The third layer would be a linear scale of $\frac{1}{3}$ with the same purpose, Layer "n" would be a linear scale of $\frac{1}{n}$ of the original image.
We will do as many loops for the optimiser as we have layers, starting with the smallest one. Then, at each layer, we initialise the parameters with the ones found in the previous layer, scaling the translation parameters accordingly.
After testing the file ***evaluation.m*** (wich we designed in order to test all the combinations of images and different metrics and techniques), the multi-resolution appeared to be not only faster but also more accurate in the registration. Here is an example of the timining and the result images (Figure 3) tested in 12 combinations.

No-Multiresolution : *288.84 sec*
Multiresolution (2 layers) : *201.36 sec*

5 Experiments & Analysis

After implementing the functionalities and code extensions that were required for this lab, we analysed how each one performed against different combinations of fixed and moving images, namely the ones provided by the lab.

In this section we can see how different strategies lead to different results. Here we present two examples: In the Figure [3] below we see that the combination of rigid transformation + normal cross correlation metric seems to work well with the images 1 and 2 (both ways), but fail to achieve a correct registration for all the other combinations. The depicted images correspond to the absolute difference of the fixed image and the registered moving image.

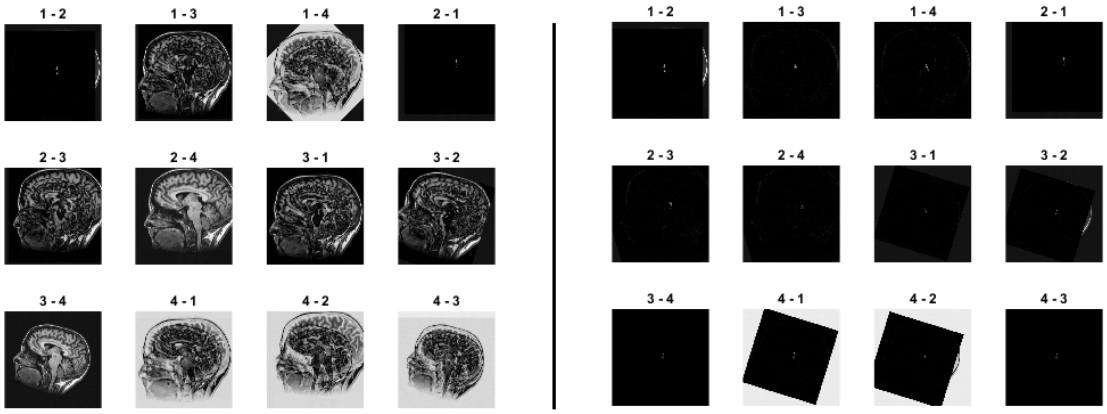

Figure 3: Comparison of two methods with all 12 combinations

In the right side of the previous Figure [3] we can see how our proposed method manages to register perfectly (in a visual level) all combinations of the given images.

This new strategy which we adopted involves two extra sequential steps (and a bonus):

- Multi -resolution
- 2 step optimizer
- Intensity inversion

The multi-resolution approach by only having 2 layers lead to results that were significantly better compared to the non multi-resolution approach. The optimization approach became a two step strategy; in the first step, the optimiser will quickly try to approach a good base result (we limit the iterations and increase the error tolerance so that this step is fast). Then, the second step is a repetition of the first, but starting with the point reached by the previous step. About the bonus: it so happens in real life that common mistakes occur, and the good news about this is that they are as easy to detect than to solve, and this is where we stepped in: we added a simple check to assess the leaning side of the histogram, so that if two images were inverted, we transformed the moving one to be able to correctly register with the fixed one. If this was not a mistake, and we have to keep the same intensities, we just have to comment the line where it happens. The robustness of this approach lies in the fact that even with extreme deformations, the leaning side of the histogram should be a clear marker for inversions.

These tests allowed us to assess in a qualitative way the performance of different parameters. However, we should also do some more precise quantitative tests, which we will see in the next section.

5.1 More Experiments & Analysis

Having tested all combinations with the initial 4 image set, we decided to put the system to the test with known transformations.

As an example, we hereby list examples of transformations applied to the images, written as the vector of parameters:

a1 = 6.0000 -3.0000 2.0000 2.5000 2.0000 3.0000
a2 = 1.0000 0.1000 -0.0000 0.1000 1.0000 0.3000

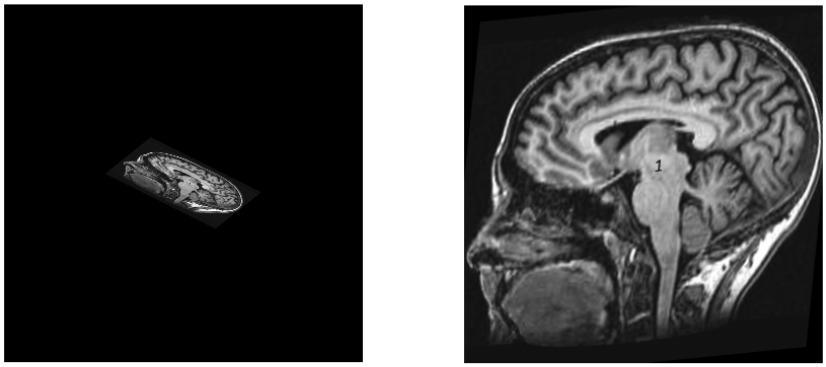

Figure 4: examples of artificially modified images with respect to the original brain 1

A good indication of the correct registration when we run the code is when we see that the iterative process stopped because it reached the desired tolerance.
In the Annex I we can see tables comparing the results between different methods against different image transformations. We assess the quality of the method by recording the execution time and the euclidean distance between the true vector and the one obtained by the optimiser.

The "Repetition" mode of the "cc" metric with "affine" transformation seems to be the best out of the other different combinations, with the multilayers adding to speed of convergence.

We also observe how sometimes, having an extra layer might get the optimiser trapped in what would appear to be a local minima, hindering the job for the subsequent layers.

6 Conclusions

After implementing the requested points for this lab, and seeing how none of the methods could work with some of the image combinations, we developed a more robust strategy that achieved perfect registration results with both the examples provided by the lab, and the ones we created.

Some of the strategies were not able to reach a perfect registration, for example using a rigid transformation. The metric used also played an important role, making it easier or more difficult for the optimiser to skip local minimas and reach the desired point.

By means of both qualitative and quantitative tests we were able to assess the performance of the various strategies. As shown in the Annex I tests, we saw that in the particular cases tested, a certain transformation and metric worked best (affine transformation and Cross Correlation as a similarity metric), and adding extra layers had the ability to speed up the process or help the optimiser get to the optimal point.

Further tests would need to be performed in a bigger scale to be able to assess the robustness of the strategy, and with real cases, since real cases are seldom affine transformations of an

original image, having perhaps body parts move with respect to fixed structures (due to atrophy, different point of view, movement of internal organs...) or the new image could be taken with a different machine, adding non-linearity to the intensities, for example.

In conclusion, during this lab we had a better understanding of the framework to register images, and were able to add extra features to the given one.

References

[1] R. O'Callaghan and T. Haga, "Robust change-detection by normalised gradient-correlation," in *2007 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1–8, IEEE, 2007.

7 Annex I

Quantitative analysis of the performance of different combinations of strategies.

Test Vector x =[1.5000 -0.5000 1.0000]					
Transformation	Metric	Layers	Repetition	Time [sec]	Euclidean distance
Rigid	sd	1	NO	3.4955	2.002
Rigid	sd	2	NO	4.2586	1.8341
Rigid	sd	3	NO	8.0984	1.8341
Rigid	sd	4	NO	5.2832	9.5147e+02
Rigid	sd	1	YES	5.2282	2.0024
Rigid	sd	2	YES	6.6706	1.8341
Rigid	sd	3	YES	7.9212	9.0000
Rigid	sd	4	YES	7.1293	9.5147e+02
Rigid	cc	1	NO	5.7737	8.9999
Rigid	cc	2	NO	12.429	1.0632e+02
Rigid	cc	3	NO	7.7958	1.0143e+03
Rigid	cc	4	NO	8.8198	1.0124e+04
Rigid	cc	1	YES	7.7902	9.0000
Rigid	cc	2	YES	15.329	1.0632e+02
Rigid	cc	3	YES	10.524	1.0143e+03
Rigid	cc	4	YES	8.8884	1.0124e+04
Rigid	gcc	1	NO	6.4732	1.4666
Rigid	gcc	2	NO	7.7065	8.9999
Rigid	gcc	3	NO	6.4687	1.0143e+03
Rigid	gcc	4	NO	6.2964	4.4882e+02
Rigid	gcc	1	YES	7.8798	1.4666
Rigid	gcc	2	YES	11.522	8.999
Rigid	gcc	3	YES	12.544	1.0143e+03
Rigid	gcc	4	YES	8.0057	1.0143e+03

Test Vector x =[1.5000 -0.5000 1.0000 -0.5000 2.0000 3.0000]					
Transformation	Metric	Layers	Repetition	Time [sec]	Euclidean distance
Affine	sd	1	NO	17.706	2.3937
Affine	sd	2	NO	17.561	0
Affine	sd	3	NO	10.713	0
Affine	sd	4	NO	8.0045	0
Affine	sd	1	YES	32.749	0
Affine	sd	2	YES	15.505	0
Affine	sd	3	YES	8.8449	9.0000
Affine	sd	4	YES	12.060	0
Affine	cc	1	NO	34.882	6.9479
Affine	cc	2	NO	14.899	0
Affine	cc	3	NO	10.815	0
Affine	cc	4	NO	7.7762	0
Affine	cc	1	YES	41.711	0
Affine	cc	2	YES	14.912	0
Affine	cc	3	YES	9.4110	1.0143e+03
Affine	cc	4	YES	12.692	0
Affine	gcc	1	NO	20.965	3.2385
Affine	gcc	2	NO	24.465	3.3168
Affine	gcc	3	NO	21.467	3.1941
Affine	gcc	4	NO	26.342	3.0993
Affine	gcc	1	YES	45.278	3.5471
Affine	gcc	2	YES	31.724	3.1657
Affine	gcc	3	YES	9.5180	1.0143e+03
Affine	gcc	4	YES	39.459	3.7223

Test Vector x =[6.0000 -3.0000 2.0000]					
Transformation	Metric	Layers	Repetition	Time [sec]	Euclidean distance
Rigid	sd	1	NO	6.3408	28.2060
Rigid	sd	2	NO	7.2816	10.5826
Rigid	sd	3	NO	5.6393	33.9150
Rigid	sd	4	NO	6.1954	5.8348e+02
Rigid	sd	1	YES	8.5332	28.4154
Rigid	sd	2	YES	9.0278	10.5836
Rigid	sd	3	YES	7.1623	36.8873
Rigid	sd	4	YES	9.7453	5.8348e+02
Rigid	cc	1	NO	10.235	10.4702
Rigid	cc	2	NO	5.7479	69.7084
Rigid	cc	3	NO	5.4319	5.8348e+02
Rigid	cc	4	NO	6.7421	3.3480e+03
Rigid	cc	1	YES	12.573	10.4702
Rigid	cc	2	YES	6.9318	69.7084
Rigid	cc	3	YES	7.7402	5.5470e+02
Rigid	cc	4	YES	8.8180	3.3480e+03
Rigid	gcc	1	NO	6.0090	6.3176
Rigid	gcc	2	NO	4.8165	7.4439
Rigid	gcc	3	NO	4.9289	18.0000
Rigid	gcc	4	NO	4.9757	2.0649e+02
Rigid	gcc	1	YES	8.4619	6.3176
Rigid	gcc	2	YES	7.3947	7.4439
Rigid	gcc	3	YES	8.5851	18.0000
Rigid	gcc	4	YES	7.5284	2.0649e+02

Test Vector x =[6.0000 -3.0000 2.0000 2.5000 2.0000 3.0000]					
Transformation	Metric	Layers	Repetition	Time [sec]	Euclidean distance
Affine	sd	1	NO	13.2157	4.9361
Affine	sd	2	NO	23.7288	4.4283
Affine	sd	3	NO	21.4318	4.4515
Affine	sd	4	NO	16.4291	4.2317
Affine	sd	1	YES	34.7076	5.0634
Affine	sd	2	YES	24.7529	4.3353
Affine	sd	3	YES	7.06022	36.8873
Affine	sd	4	YES	40.7985	5.5899
Affine	cc	1	NO	21.6985	4.7049
Affine	cc	2	NO	17.5722	0
Affine	cc	3	NO	20.9592	0
Affine	cc	4	NO	21.5462	24.7737
Affine	cc	1	YES	21.9481	0
Affine	cc	2	YES	16.4350	0
Affine	cc	3	YES	7.66995	0
Affine	cc	4	YES	27.8599	0
Affine	gcc	1	NO	22.2441	6.2263
Affine	gcc	2	NO	13.8488	7.3413
Affine	gcc	3	NO	21.1957	6.4414
Affine	gcc	4	NO	19.5221	6.7193
Affine	gcc	1	YES	19.6742	6.2408
Affine	gcc	2	YES	31.4452	7.4190
Affine	gcc	3	YES	10.0361	18.000
Affine	gcc	4	YES	27.5418	6.2408

Test Vector x =[2.0000 -2.0000 0.0000]					
Transformation	Metric	Layers	Repetition	Time [sec]	Euclidean distance
Rigid	sd	1	NO	5.6014	0
Rigid	sd	2	NO	3.4025	0
Rigid	sd	3	NO	4.1418	0
Rigid	sd	4	NO	3.5996	0
Rigid	sd	1	YES	7.7512	0
Rigid	sd	2	YES	5.5178	0
Rigid	sd	3	YES	4.3237	0
Rigid	sd	4	YES	4.3224	0
Rigid	cc	1	NO	3.9071	0
Rigid	cc	2	NO	3.3440	0
Rigid	cc	3	NO	3.5018	0
Rigid	cc	4	NO	3.7195	0
Rigid	cc	1	YES	5.1555	0
Rigid	cc	2	YES	4.6793	0
Rigid	cc	3	YES	4.8768	0
Rigid	cc	4	YES	5.0200	0
Rigid	gcc	1	NO	8.9857	0
Rigid	gcc	2	NO	4.3352	0
Rigid	gcc	3	NO	3.9834	0
Rigid	gcc	4	NO	3.5025	0
Rigid	gcc	1	YES	20.2776	0
Rigid	gcc	2	YES	12.2328	0
Rigid	gcc	3	YES	11.7187	0
Rigid	gcc	4	YES	12.4369	0

Test Vector x =[2.0000 -2.0000 0.0000 2.5000 0.5000 0.5000]					
Transformation	Metric	Layers	Repetition	Time [sec]	Euclidean distance
Affine	sd	1	NO	33.0389	5.2939
Affine	sd	2	NO	29.2282	4.1751
Affine	sd	3	NO	26.9016	9.3714
Affine	sd	4	NO	24.0563	5.0698
Affine	sd	1	YES	38.1334	5.5657
Affine	sd	2	YES	43.5109	5.6574
Affine	sd	3	YES	4.3602	0
Affine	sd	4	YES	40.2891	6.9417
Affine	cc	1	NO	15.5678	3.4290
Affine	cc	2	NO	27.2459	9.4769
Affine	cc	3	NO	18.6423	5.0931
Affine	cc	4	NO	23.2428	8.1105
Affine	cc	1	YES	30.8729	0
Affine	cc	2	YES	36.8633	6.9277
Affine	cc	3	YES	6.8192	0
Affine	cc	4	YES	34.7618	7.9775
Affine	gcc	1	NO	14.4532	3.6435
Affine	gcc	2	NO	30.8782	3.6196
Affine	gcc	3	NO	21.8378	3.9608
Affine	gcc	4	NO	46.0078	3.7498
Affine	gcc	1	YES	77.7586	3.6364
Affine	gcc	2	YES	99.8787	4.5088
Affine	gcc	3	YES	12.5878	0
Affine	gcc	4	YES	61.6084	3.6364